A Taste of Reactive Systems

Michael Christensen

July 2, 2018

Reactive Systems

Systems which continously interact with physical environment ("The Synchronous Data Flow Programming Language LUSTRE" Halbwachs et al. 1991)

Dataflow

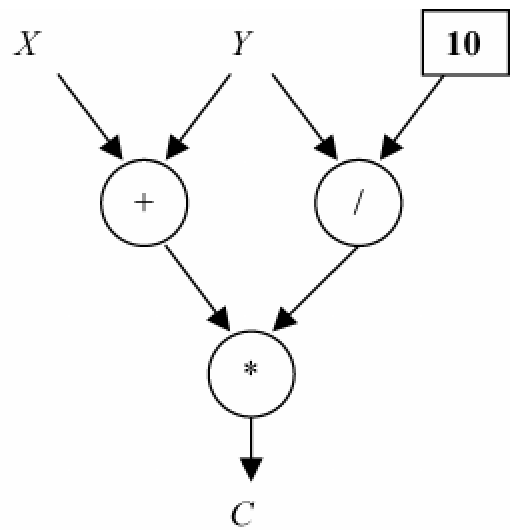
Dataflow

- Program is a directed graph
- **Nodes** are primitive instructions
- Directed **arcs** are data dependencies
- Flow along arcs like unbounded FIFO queue
- Initially for exploiting parallelism

Example

$A := X + Y$
$B := Y / 10$
$C := A * B$

(a)



(b)

Implementation Approaches

- Data Availability-Driven (*push*)
  - efficient and low-latency (good for RTS)
- Demand-Driven (*pull*)
  - eliminate unneeded nodes, flexible

Iteration

- Loop body variables have **same** throughout iteration
- Variables updated with `NEW` operator, e.g. `NEW X = X + 1`

Data Structures

- "I-structures" for making undefined data immediately available

Languages

- Freedom from side effects
- Data dependency $=$ scheduling
- Single assignment of variables
- E.g. TDFL, LAU, Lucid, Id

Visual

- Let user see and manipulate program graph
- E.g. DDNs, GPL, LabView, ProGraph, NL

Synchronous Dataflow (see next section)

- Number of tokens consumed/produced on each arc is known at compile-time
- Statically schedulable

Synchronous Languages

Synchronous Languages

- Notion of clock as first-class values
- Program (i.e "reaction") is conjunction of reactions for each block and connections between blocks
- Languages differ in how they deal with parallel composition constraints

Lustre

`x = y + z` at each instant `k`, $\mathrm{x}_k = \mathrm{y}_k + \mathrm{z}_k$

- Declarative
- Each variable is a function of discrete time
- Variables are 'flows' (infinite sequences of values)
- Operators extended **pointwise** over flows
- **temporal** operators for describing sequential flow:
    - `pre(x)`, `->`, `when`, `current`
- Structured programming via **nodes**
    - function over typed input flows producing output flows

Lustre (continued)

- Activate different program parts at different rates via **clocks**
  - basic clock is finest notion of time (external)
  - create slower clocks from basic clock, others
  - `x when c`
  - operators operate on same-clock flows

Lustre Example

```
node COUNT (init, incr: int, reset: bool)
  returns (n: int):
let
  n = init ->
          if reset then init else pre(n) + incr;
tel

node SIMPLE_STOPWATCH
     (start_stop, reset, hs: bool)
         returns (time: int);
var CK, running: bool;
let
  time =
    current(COUNT((0, 1, reset) when CK));
  CK = true -> (HS and running) or reset;
  running =
    TWO_STATES(false, start_stop, start_stop);
tel;
```

Esterel

- Imperative for describing control
- Program is a set of nested *concurrently running threads*, synchronized on single global clock
- On reaction start, thread resumes from `pause` statement
- Threads communicate via global events ("signals")
- Preemption statements (`abort`) tests predicate before reaction

Esterel (continued)

Example

```
loop
    suspend
        await Play; emit Change
    when Locked;
    abort
        run CodeForPlay
    when Change
end
||
loop
    suspend
        await Stop; emit Change
    when Locked;
    abort
        run CodeForStop
    when Change
end
||
every Lock do
    abort
        sustain Locked
    when Unlock
end
```

Synchronous Conclusions

- Other languages
  - Statecharts
  - Signal
- Discussion of
  - industrial application
  - compilation techniques
    - Esterel: automate-based
    - Signal: solve program abstraction described by clock and causality calculus
- Functional concurrency allows deployment without an OS scheduler

Reactive Programming

Reactive Programming

- Abstractions so programs are **reactions** to external events
- **Language manages** flow of time and state change propagation
- Based on synchronous dataflow with relaxed real-time constraints
    - Behaviors: continuous time-varying values (e.g. time)
    - Events: potentially infinte streams of value changes ocurring at **discrete** points in time (e.g. button press)
    - Supports higher-order dataflow
    - Supports dynamic structure
    - *Glitches* and *lifting*

Three Types of Reactive Languages

Functional Reactive

- Provides behaviors, events, event and switching combinators
- Declarative
- Temperature example
- E.g. Fran, Flapjax, Scala.react

RP Cousins

- Time-varying abstractions not integrated with rest of lang
- E.g. Cells, Trellis

Synchronous, Dataflow, and Sync DF

- Also "Real-time FRP", "Event-driven FRP"